

DeviceAdapterHClib

A VTK-m device adapter that supports the HClib runtime

Avadh Yadav, Burhan Nabi
{avadh14026, burhan14031}@iiitd.ac.in

1 Motivation

Advancement in processor technology include ever greater numbers of cores, hyperthreading, accelerators with integrated blocks of cores, and special vectorized instructions, all of which require more software parallelism to achieve peak performance. Traditional visualization solutions cannot support this extreme level of concurrency. Thus VTK-m was created to address this issue: a visualization toolkit for multi-core architectures.

VTK-m¹ does this by using the abstract device model. By using this model instead of writing the same algorithms for different devices, we can reduce the amount of software written to a feasible level. Different devices use, what is known as, a device-adapter to provide support of these algorithms for a particular device. Hence, one only needs to write a device-adapter to support the algorithms provided by VTK-m on a particular device, instead of going through the arduous task of implementing all the algorithms in a way that the device supports them.

HClib² is a task-based parallel programming model that supports the finish-async, parallel-for, and future-promise parallel programming patterns through both C and C++ APIs. The HClib runtime is a lightweight, work-stealing, and locality-aware runtime.

Our aim is to provide a device adapter that supports the HClib task parallel runtime.

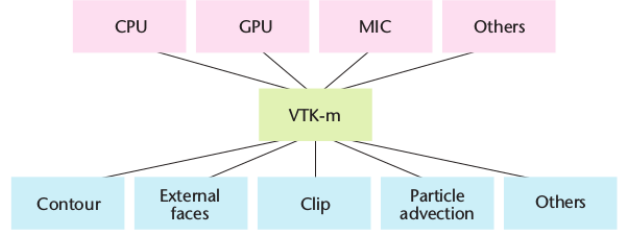


Figure 1: VTK-m abstract device model. Purple denotes devices. Pink denotes filters.

2 Problem Statement

Writing a device adapter for VTK-m that supports HClib involves implementing the following^{3 4}:

- **ArrayManagerExecutionHClib.h:** The implementation for ArrayManagerExecution is expected to manage the resources for a single array.
- **DeviceAdapterAlgorithmHClib.h :** It provides a set of algorithms that can be invoked in the control environment and are run on the execution environment.
- **DeviceAdapterTagHClib.h :** A device is identified by a device-dapter tag.
- **VirtualObjectTransferHClib.h :** This is responsible for instantiating virtual objects in the execution environment
- **DeviceAdapterHClib.h :** A trivial header that includes the above mentioned files.

¹<http://m.vtk.org/index.php/MainPage>

²<https://github.com/habanero-rice/hclib>

³These are all header files because they use templated types, in accordance with the VTK-m design

⁴<http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf>

⁵<http://www.openmp.org/>

⁶<https://www.threadingbuildingblocks.org/>

3 Literature

3.1 VtkSMP

This paper focuses on task based parallel programming environments, and delimits potential parallelism through tasks. It supports three different environments namely : *OpenMP*⁵, *Intel TBB*⁶ and *X-KA-API*⁷. It suggests three different parallelization patterns that can be reused in various VTK filters:

- The first pattern targets loops with independent iterations (foreach loop) producing independent data chunks of known size that can be directly written in a global data structure without concurrency related issues.
- A strategy to handle foreach loops producing data with unknown memory footprints are also addressed by this paper.
- A parallelized tree traversal, a pattern relevant for several acceleration data structures is also suggested by the paper.

3.2 VTK-m

VTK-m⁸ used data parallel primitives to provide an abstraction layer between the low-level hardware architecture and the high-level code, which both increases the portability of VTK-m and simplifies the implementation of algorithms.

VTK-m does this by using the abstract device model. By using this model instead of writing the same algorithms for different devices, we can reduce the amount of software written to a feasible level. Different devices use, what is known as, a device-adapter to provide support of these algorithms for a particular device. Hence, one only needs to write a device-adapter to support the algorithms provided by VTK-m on a particular device, instead of going through the arduous task of implementing all the algorithms in a way that the device supports them.

3.3 Nvidia Index Plug-in for ParaView

*Nvidia Index*⁹ leverages GPU-clusters for scalable large-scale data visualization. It gives a scalable GPU-accelerated solution for high-quality visualization of large volumetric and surface data as well as interactive visualizations. The plug-in provides an

infrastructure for the creation of rich visualization interaction techniques.

3.4 Optimizing File Access Patterns through the Spatio-Temporal Pipeline for Parallel Visualization and Analysis

The main bottleneck in large-scale parallel analysis is usually the I/O read step. One of the main factors in I/O performance is how the file is accessed, and what the read pattern is. The parallel decomposition strategy determines the read pattern. This paper introduces a model in which we can estimate the I/O read time for a file, given the partitioning of the file. Using this model we were able to configure the spatio-temporal pipeline to use read patterns which obtained greater I/O performance versus read patterns produced by the more common method of spatial parallelism.

Most of the common visualization tools either uses spatial parallelism or temporal parallelism. In spatial parallelism, the data is partitioned spatially and spread out across several processes. Each process then applies the same computation on their piece of data. Whereas Temporal parallelism involves processing data from different timesteps simultaneously. The same computations are applied to each timestep.

This paper uses spatio-temporal pipeline to utilize both spatial and temporal parallelism, which allows for better control of the access pattern used to read files. Within the spatio-temporal pipeline, all available processes are divided into groups called time compartments. Temporal parallelism is utilized as different timesteps are independently processed by separate time compartments, and spatial parallelism is used to divide each timestep over all processes within a time compartment.

The ratio between spatial and temporal parallelism is controlled by adjusting the size of a time compartment. Assuming the number of total processes is constant, if the time compartment size is large, there is lower temporal parallelism and higher spatial parallelism. If the size of a time compartment is lowered, there is higher temporal parallelism and lower spatial parallelism.

⁷<http://kaapi.gforge.inria.fr/index.md>

⁸<http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf>

⁹<https://developer.nvidia.com/index>

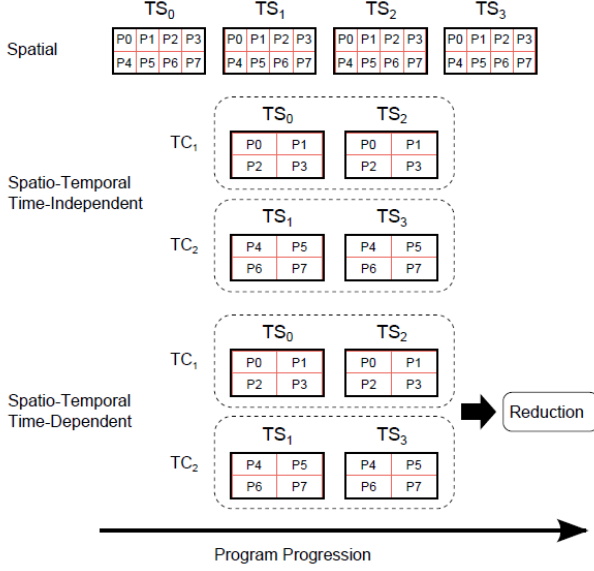


Figure 2: Spatial parallelism and the spatio-temporal pipeline. Eight processes are used for four timesteps. Timestep i , TS_i , is represented by a black rectangle. Red rectangles indicate spatial partitions, with the label P_i indicating which process gets that partition. TC_1 and TC_2 are the two time compartments used. (Top) Using only spatial parallelism creates eight partitions per file, and files are processed serially. (Middle) The time-independent use case for the spatio-temporal pipeline. Eight processes are split into two time compartments. Each time compartment processes two timesteps. (Bottom) The time-dependent use case for the spatio-temporal pipeline. A reduction step is performed after all timesteps have been processed.

On performing several timing test, the file access patterns resulting from the spatio-temporal pipeline achieved over a factor of more than 400 speedup over the read patterns used by the spatial method.

4 Solution

All the source code related to HCLib Device Adapter is placed in the sub-directory of `vtkm/cont` as `vtkm/cont/hclib`. By convention the implementation of device adapters within VTK-m are divided into 4 header files with the names

`DeviceAdapterTag.h`, `ArrayManagerExecution.h`, `VirtualObjectTransfer.h`, and `DeviceAdapterAlgorithm.h`, which are hidden in internal directories. The `DeviceAdapter.h` that most code includes is a trivial header that simply includes these 4 files. The functions of these files are already mentioned earlier.

We implemented the device-adapter which can be found here ¹⁰. The device adapter has schedule algorithms with HCLib support as well parallelized versions of some commonly used algorithms like Scans etc. There is a lot of scope for improvement.

5 Appendix

5.1 Installation

VTK¹¹ We started out by installing VTK and playing with it. The installation of VTK was mostly straightforward. At the time we did not know that Qt was a requirement for some of the examples in VTK, and hence did not install it. But we were able to run examples without dependencies easily enough. To install:

```
$ git clone git://vtk.org/VTK.git
$ cd VTK

$ mkdir ~/VTK-build
$ cd ~/VTK-build

$ cmake ../VTK
```

You should see something like this ¹². Choose the configuration that you want to generate. Then build using **make**.

Paraview¹³ Installation of Paraview was tricky and the vanilla installation failed many times without any descriptive error messages. This happened mostly because of issues with dependencies. The *SuperBuild*¹⁴ installation of paraview was not working at first, but then the most basic configuration of selecting only MPI and Qt worked and we were able to install Paraview using SuperBuild. ¹⁵

Paraview installation using Superbuild is quite straight-forward. It downloads all the dependencies we have configured before building it on its own.

¹⁰<https://gitlab.com/chiranjibsur/vtkExamples/tree/master/VTKm/hclib/device-adapter/hclib>

¹¹http://www.vtk.org/Wiki/VTK/Configure_and_Build

¹²<http://www.vtk.org/Wiki/File:Vtk-cmake.png>

¹³https://www.paraview.org/Wiki/ParaView:Build_and_install

¹⁴<https://gitlab.kitware.com/paraview/paraview-superbuild/>

¹⁵The installation of Paraview takes quite some time and the best way to install it would be using SuperBuild.

```
$ git clone --recursive
https://gitlab.kitware.com/paraview/paraview-super\
  build.git
$ cd paraview-superbuild
$ git fetch origin
$ git checkout v5.2.0
$ git submodule update
```

VTK-m¹⁶ The major issues while installing VTK-m were getting the dependencies to work. Installation of TBB from source and then exporting the required directories posed problems in installation and did not work. Only when TBB was installed using the package-handle¹⁷ were we able to install VTK-m.

MPI Download MPI from OpenMPI website:

```
$ gunzip -c openmpi-2.1.1.tar.gz | tar xf -
$ cd openmpi-2.1.1
$ ./configure --prefix=/usr/local
$ make all install
```

CMake¹⁸ CMake can be build by following instruction on the website and does not pose much problems. Avoid installation using the package manager, as it would normally be some versions behind the current stable build and may lead to problems. Ensure that the command **ccmake** works as well after installing it as it is commonly used in installation of the above three.

5.2 Working with examples

All of the above use CMake as a builder and packaging tool. To work with examples CMake is required. The best way to build example using cmake is to follow the following steps(see figure 2):

- Change the directory to the directory where the example is located.
- Make a directory called build and go to it.
- Run cmake by passing the parent directory(where the example is located) as the argument.
- If CMake runs successfully, it should produce a **Makefile**, which can then be used to generate executables in a normal fashion.
- Cleaning up is as simple as removing the build directory.

5.3 Difference between VTK/VTK-m and Paraview

VTK/VTK-m are libraries that applications can use. They are used to process, render data and produce visualization that provide meaning.

Paraview, on the other hand, is a GUI application that you can fire up and start looking at your data. It was meant to be easy to use even for non-programmers and its primary goal is to be able to

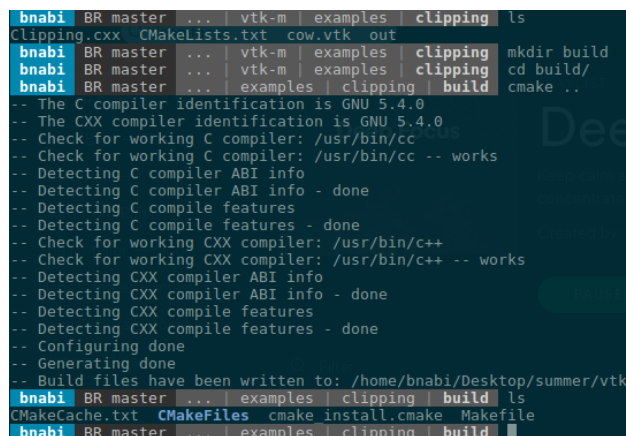


Figure 3: An illustration of building an example

scale from netbooks to worlds largest computers. It used the above mentioned libraries in the background.

5.4 Running examples which require input files

Some examples in VTK-m require input files e.g. the Clippings example. On running the executable without any input files, one will get the input message: Usage:

```
$ ./Clippings_SERIAL <input_vtk_file>
[fieldName] <isoval> <output_vtk_file>
```

Value of fieldName is found inside the input files So, the command could be something like:

```
$ ./Clippings_SERIAL cow.vtk fixed 0.5 output
```

¹⁶http://m.vtk.org/index.php/Building_VTK-m

¹⁷ Using `sudo apt-get install libtbb-dev` (for Debian/Ubuntu based system) to install TBB

¹⁸<https://cmake.org/>

5.5 Writing CMakeLists.txt¹⁹ files

When starting to write new examples, or tinkering with the existing ones, one may need to write a custom *CMakeLists* which is basically a kind of a MakeFile to specify the executable and its dependencies. The file below illustrates such a *CMakeLists* file ²⁰ :

```
find_package(VTKm REQUIRED
  COMPONENTS Serial OpenGL Rendering GLUT
  OPTIONAL_COMPONENTS TBB
)

add_executable( example example.cxx)

target_include_directories( example PRIVATE
  ${VTKm_INCLUDE_DIRS})
target_link_libraries(example ${VTKm_LIBRARIES})
target_compile_options(example PRIVATE
  ${VTKm_COMPILE_OPTIONS})
```

5.6 Integrating HClib

Habanero-C is a compiler-free light-weight standalone work-stealing library. It is under development in the Habanero project at Rice University. It serves as a research testbed for new compiler and runtime software technologies for extreme scale systems for homogeneous and heterogeneous processors. The Habanero-C language extends the C language with parallel constructs which include, forasync , async, finish, next, single , isolated, futures, at and await to help achieve asynchronous fine grain task parallelism.

HClib can be downloaded and set up using the instructions given on the github page here ²¹. Thanks to the ease with which CMake lets us integrate different dependencies, getting HClib integrated with VTK-m was not a big issue, and examples of that can be seen here ²².

5.7 Cmake can't find QT5 library

Look at where you installed QT5 and create an environment variable pointing to it.

```
export
CMAKE_PREFIX_PATH=/home/bnabi/Qt/5.8/gcc_64/;
```

5.8 Leveraging parallel processing in Paraview

To leverage parallel processing capabilities in paraview or pvpython, one has to use remote visualization, i.e., one has to connect to a pvserver. We can start pvserver to run on more than one processing core using mpirun.

```
mpirun -np 4 pvserver
```

The client can be run using the normally. The client has to be connected to the server before use.

5.9 CMake error when trying to build VTK-m examples

This can be resolved by specifying a minimum version of cmake as:

```
cmake_minimum_required (VERSION 2.6)
```

5.10 Tip on searching

While working, you might need to look search all source files for a certain pattern e.g. one might need to see all the files in which MPI is used. One could use grep for it, running the given command from the root of the directory in which we want to look for the patter:

```
grep -R <pattern> *
```

5.11 Configuring vtk, paraview, vtk-m

Installation of all of the above require configuraton using CMake. This can be achieved using the following steps:

Iterative process

Select values, run configure (c key).

Set the settings

Repeat until all values are set.

Generate option is available (g key).

Some variables (advanced variables) are **not** visible right away.

To see advanced variables, toggle to advanced mode (t key).

¹⁹<https://cmake.org/cmake-tutorial/>

²⁰Writing a CMakeLists file is not that difficult, documentation should suffice

²¹<https://github.com/habanero-rice/hclib>

²²<https://gitlab.com/chiranjibsur/vtkExamples/tree/master/VTKm/hclib>

Once you're done type in the make command. You can use make.

```
make -j<N>
N: number of cores
```

5.12 Running examples using a different device adapter

After writing a custom CMakeLists.txt file and surround the example with hclib::launch tag. One can compile the normally, by first running cmake and the make.

A simple example can be found here ²³.

Here is the output on running the MarchingCubes filter with the HCLib device-adapter:

```
bnabi ... | report | demo | build 130 | make -j2
Scanning dependencies of target Demo
[ 50%] Building CXX object CMakeFiles/Demo.dir/Demo.cxx.o
[100%] Linking CXX executable Demo
[100%] Built target Demo
bnabi ... | report | demo | build 130 | ./Demo
Device Adapter Name: HCLib
-----HCLIB_RUNTIME_INFO-----
>>> HCLIB_WORKERS      = (null)
>>> HCLIB_HPT_FILE     = (null)
>>> HCLIB_BIND_THREADS = false
>>> HCLIB_STATS        = 1
-----
WARNING: HCLIB_WORKERS not provided, running with default of 4
Using 4 worker threads (including main thread)
about to render the results of the MarchingCubes filter
===== HCLIB Runtime Statistics =====
totalAsyncTasks totalAsyncStolen
2                2
Total time: 3409.088 ms
----- End HCLIB Statistics -----
===== TEST PASSED in 3409.088 msec =====
bnabi ... | report | demo | build 130
```

Figure 4: Running example with a diff device adapter

6 Conclusion

We have created a device adapter for HCLib and any system having HCLib installed can successfully run

it. But we believe that there is more scope of improvement in parallelism which we might see in the upcoming versions of VTK-m since as of now only sort, scan filters have been parallelized.

Also apart from that a separate device adapter can be created which can support hybrid parallelism.

7 References

- <http://m.vtk.org/index.php/MainPage>
- <https://github.com/habanero-rice/hclib>
- <http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf>
- <http://www.openmp.org/>
- <https://www.threadingbuildingblocks.org/>
- <http://kaapi.gforge.inria.fr/index.md>
- <http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf>
- <https://developer.nvidia.com/index>
- <https://gitlab.com/chiranjibsur/vtkExamples/tree/master/VTKm/report/demo>
- <https://github.com/habanero-rice/hclib>
- <http://www.vtk.org/Wiki/VTK/Con>
- <http://www.vtk.org/Wiki/File:Vtk-ccmake.png>
- <https://www.paraview.org/Wiki/ParaView:BuildAndInstall>
- <https://gitlab.kitware.com/paraview/paraview-superbuild/>
- <http://m.vtk.org/index.php/BuildingVTK-m>
- <https://cmake.org/>
- <https://cmake.org/cmake-tutorial/>
- <https://github.com/habanero-rice/hclib>
- <https://gitlab.com/chiranjibsur/vtkExamples/tree/master/VTKm/report/demo>
- <https://gitlab.com/chiranjibsur/vtkExamples/tree/master/VTKm/report/demo>

²³<https://gitlab.com/chiranjibsur/vtkExamples/tree/master/VTKm/report/demo>